Decision Process in Human-Agent Interaction: extending Jason Reasoning Cycle

Cite this paper as:

- Chella A., Lanza F., Seidita V. (2019) Decision Process in Human-Agent Interaction: Extending Jason Reasoning Cycle. In: Weyns D., Mascardi V., Ricci A. (eds) Engineering Multi-Agent Systems. EMAS 2018. Lecture Notes in Computer Science, vol 11375. Springer, Cham

Antonio Chella
Dipartimento di Ingegneria
antonio.chella@unipa.it

Francesco Lanza
Dipartimento di Ingegneria
francesco.lanza@unipa.it

Valeria Seidita
Dipartimento di Ingegneria
valeria.seidita@unipa.it

Abstract

The main characteristic of an agent is acting on behalf of humans. Then, agents are employed as modeling paradigms for complex systems and their implementation. Today we are witnessing a growing increase in systems complexity, mainly when the presence of human beings and their interactions with the system introduces a dynamic variable not easily manageable during design phases. Design and implementation of this type of systems highlight the problem of making the system able to decide in autonomy. In this work we propose an implementation, based on Jason, of a cognitive architecture whose modules allow structuring the decision-making process by the internal states of the agents, thus combining aspects of self-modeling and theory of the mind.

Keywords

Human-Agent Interaction; BDI Agent; Jason.

1 Introduction

Today we want software able to cooperate with us, to anticipate our needs and to coordinate its activities with us. We also wish to have software that can autonomously and intelligently intervene and act in dynamic and changing contexts, operating as humans would do. For example, an agent-human team has to cooperate to achieve a common goal in an environment not fully known. All the members of the team have to decompose the overall goal into a series of subgoals. They should then be able to understand or learn which actions are needed to reach the objective. Finally, they should match their skills with the correct steps to perform, and eventually, they should delegate some task to each other. This scenario concerns fully autonomous cooperative work that requires a complex software system with runtime adaptation to new situations that may lead to new requirements and constraints. Everything injected and evaluated at runtime cannot be defined during design phases, and therefore the system has to be handled as a self-adaptive system. In brief, a self-adaptive system must be aware of its objectives; it must be able to monitor the working environment and understand how far it is and if it is deviating from the objective. Moreover, it must be able to adopt alternative plans and it must also be able to generate new plans when necessary.

Important challenges in this field concern knowledge representation and updating, the selection and creation of plans at runtime, the invention of techniques for purposefully and efficiently conveying the (runtime) decision process. These challenges lead to different solutions depending on whether we look at the architectural level or the system level. At the architectural level, it is necessary to identify a set of cognitive modules for modeling the cognitive process behind decisions in the before said scenario. At the system level, it is necessary to employ the right technological solution for coding and implementing a system working in changing conditions and in continuous interaction with the human.

In this chapter, we focus on the system level counterpart of the decision process that we achieve by employing BDI agents paradigm [1] and Jason as an agent language [2][3]. Decision processes elaborate data coming from external sources and the environment. In many domains, it would not be enough, or it would be hard to design and implement the decision process merely employing the monitoring, analyzing, planning, acting (MAPE) cycle [4]. In our view, the decision process should take as input all the internal states of agents involved in the environment, including human. Internal states then embody the changes occurring at runtime. The work we discuss aims at considering, as a crucial part of the decision process, the data coming from the capability of attributing mental states (beliefs, desires, emotions, knowledge, abilities) to itself and the other. In brief, we take into account self-modeling and theory of mind capabilities. Also, we briefly illustrate the architectural level in the form of the cognitive architecture we identified to include modules for knowledge representation and management, for internal states modeling and for the decision process.

Contribution and Outline of the chapter. In this chapter, we illustrate the first steps of our ongoing work aiming at integrating self-modeling and the theory of mind in an architectural structure to implement an adaptive decision process at the architectural and the system level. The architectural part extends the MAPE cycle with modules allowing the perception of the external and the inner world in the form of internal states. The way we structured the architectural part and the rationale it underpins let us quickly fill the gap with the system level. We then present the core of this chapter, an extended version of the Jason reasoning cycle to map the architectural level into an agent framework.

The chapter is organized as follows. Section 2 discusses the features and challenges of human-agent interaction; in Section 3 we briefly describe the architectural level of the proposed work; in Section 4 we illustrate Jason features and its reasoning cycle and in Section 5 we show how we extend Jason reasoning to implement the cognitive architecture and its decision process modules at the system level. Finally, Section 6 draws some discussions and conclusions.

2 Human-Agent Interaction. Features and Challenges

Agent paradigm [5] has been used since decades as a solution, both from a theoretical and an implementation point of view, for systems providing aid to humans in their everyday life. An agent is thought and programmed to act on behalf of humans. Cases studied and implemented since now refer to automated

systems emulating in some way the autonomous behavior of humans involved in complex tasks. However, automation is very different from autonomy; real autonomy also implies self-adaptation and self-organization abilities.

Autonomy intervenes in scenarios where agents have to intelligently interact with humans and with other agents to reach a common objective in an unknown or partially known environment. The environment may be not totally known at design time or may change during agents' working. A possible scenario is a human-agent team. Normally, teammates share the same objectives and the same knowledge on the environment. They know the goal and have a plan to reach it, collaborating and cooperating with the others. Collaboration and cooperation amount to knowing own abilities and others' ones, interacting each time something is unknown or is going wrong or each time a task has to be delegated to another or adopted by another. Also, collaboration and cooperation imply explaining why an action/task has failed or cannot be done. Moreover, the teammate may anticipate actions of another member as the result of continuously observing the other and the environment. Knowledge of the environment is continuously updated to let teammates re-plan or create new plans if necessary. Knowledge on the environment also includes the ability to develop a model of the self, the inner state of the teammate changes over time and influences, along with the general knowledge on the environment and on the goal to pursue, the decision-making process. We also consider that the result of anticipating actions, both on the inner and outer world, greatly affects the decision process of the action to perform.

Generalizing, features of human-agent team systems require to investigate and analyze *(i)* knowledge acquisition, representation and updating, including memory management, *(ii)* environment representation, inner and outer one, *(iii)* plans selection or creation *(iv)* learning, *(v)* introspection, for allowing team members to be aware of himself and his capabilities.

Human-agent interactions can be encoded from simple situations where everything may be identified and established at design time (environment, plans, actions and changing situations) to more complex ones where changes occur at any time and where the agent has to decide autonomously and self-adapt.

To better explain the case we are facing, let us consider the following three scenarios. In the first case, a team composed of agents and humans working together to carry out a task known to both. Let's suppose that the working environment is known in advance and during runtime, there are no changes other than those resulting from the actions of some pre-programmed agents. In this situation, agents may act in complete autonomy, and goals may be achieved performing actions in the repertory. The collaboration is apparent in the sense the agents and humans do not need mutual help; the BDI logic [6] and its implementation using Jason [2] are perfectly efficient and usable.

In the second case, let us suppose the agent needs collaboration by a human for performing some part of the overall goal. For instance, the agent may realize to be not able to do an action for some physic limitations even though having the know-how for doing the action. This situation implies an intervention of the human under an explicit request of the agent. It is a collaborative work. This case requires some "soft" self-adaptation; the agent has to be endowed with the capability to understand when it cannot select/perform an action to achieve a goal. This case can also be handled with the use of the Jason interpreter by customizing methods of some of its predefined classes (see [2] for more details).

In the third case, the most complicated one, let us suppose that only part of the environment is known beforehand. The common goal, as well as a set of plans to achieve it, are identified at design time, but the ongoing interaction of the agent with the environment and with the human change conditions unpredictably. This fact happens when interactions bring out new terms of operability that must be worked out to decide what action to take. Generally, when a team is made up of only humans, they choose actions from their experience, the knowledge they have of the other team member, the trust they place in the other team, their emotional state or the anticipation of the actions of others.
For example, suppose that two people are caring for a disabled patient. Routine care includes the administration of medicines, cleaning, help during meals, etc. and each of the two has been assigned tasks. Suppose that during lunch the patient spills a glass of water. In this case, it is required picking up the glass from the ground and cleaning the patient, but none of the two actions have been assigned to a specific caregiver it may happen that if one takes the initiative and decides to clean the patient, then the other chooses to pick up the glass. Moreover, if there is no procedure to respond to an emergency,

the two caregivers generally do not stay still, they decide what to do based on their experience and in general on their internal state. Besides, they will collaborate, even delegating to each other what to do. Our long-term work is to replicate this way of behaving in human-(multi) agent systems.

A human-agent team is a complex system [7], designing and developing different levels of cooperation and self-adaptation are the main challenges in this context. The system has to be equipped with the ability to select the best action to perform, to conduct the right decision-making process basing on goals, capabilities, mental state and so on not totally known at design time.

Actually, designers have not tools to analyze and identify all the possible elements that cause perturbation and change in the environment nor to implement an efficient system level (the running multi-agent system). They cannot determine the decision-making process to be implemented at the system level.

There not exist design approaches handling cooperative and self-adaptive systems where requirements and goals change at runtime, during systems working, and where systems configuration and features are the results of the interaction system, environment and humans; the system behavior emerges from interactions. Thus, a gap still exists between design and implementation level of such a kind of systems. We claim that this gap may be filled by employing theory coming from the cognitive process area: the *cognitivist* and the *emergent* system approaches [8][9][10]. The first approach relies on the common perceive-decide-act loop and on symbolic representations to physically instantiate code operations devoted to realize and implement cognitive agents behavior and decision process. The emergent system approach considers cognition as a dynamic emergent process implying self-organization. Many emergent approaches consider anticipative skills more important than knowledge acquisition and consider the physical instantiation of the cognitive model as an important factor. Issues related to the research area we are investigating may be faced using a cognitive architecture relying on the principles of an emergent approach. However, we believe that separating knowledge acquisition from anticipative skills is not the right direction. Thus, we are moving towards a hybrid model form that will be illustrated in the following section.

It is necessary to have means for analyzing how the mind builds a model of self and uses the theory of mind. Our proposal is to integrate self-modeling and theory of mind in the modules of a cognitive architecture to implement all the human-human team features handled by a multi-agent system. The work focuses both on theoretical aspects and on low-level ones. Indeed, we aim at identifying an abstract cognitive model (see section 3) and the related implementation counterpart (see section 5). In the literature, some promising approaches [11][12] propose to solve this problem by shifting the design time to runtime. Also, some architectures containing modules for learning and memory have been introduced to pass the decision-making process through the stored and processed sensing data [13][14][15]. However, these approaches do not take into account the use of mental states and cognitive processes, which are the primary element in our hypothesis to be able to create human-agent interaction systems behaving as human-human systems.

We focus on the multi-agent system managing interactions with humans in a human-like fashion. We employ the multi-agent systems paradigm for implementing our cognitive architecture; each *module* becomes an *agent* which interacts with all the others for achieving its objectives and at the same time the overall system objective. In particular, we pay attention to the *Belief-Desire-Intention model* (BDI) [6] for modeling the cognitive reasoning process of each agent.

3 The Architectural Level for the Decision Process

To realize an emergent cognitive approach and conceive a cognitive architecture useful for our purposes, we took inspiration from the standard model of the mind [16]. The standard model of the mind has been developed along three different levels, from the purely biological level of mind to the more complex deliberative one. Hence, from simple behavior to complex one. The standard model has been conceived in a way that allows extending modules in each level; the higher levels are constrained by the lower ones. The scenario we are studying lays at the higher level, the one ruling an intelligent and deliberative behavior.

The standard model is intended to be a reference model and a theoretical driving approach. It has been based on three well known cognitive architectures (ACT-R, SOAR and Sigma [17][14][18]) and presents a core composed of the cognitive cycle: perception, working memory and action. ACT-

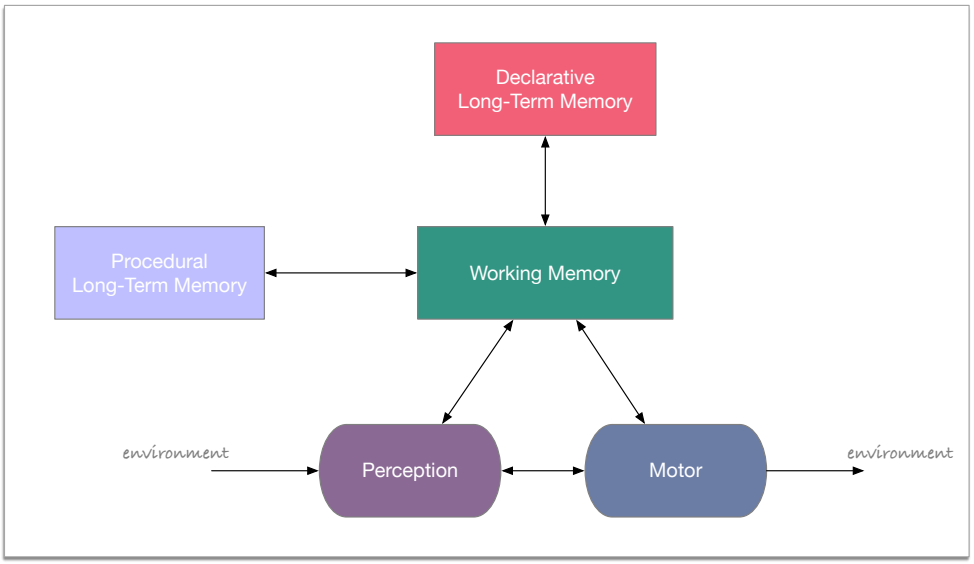

Figure 1: The Standard Model of the Mind - redrawn from [16]

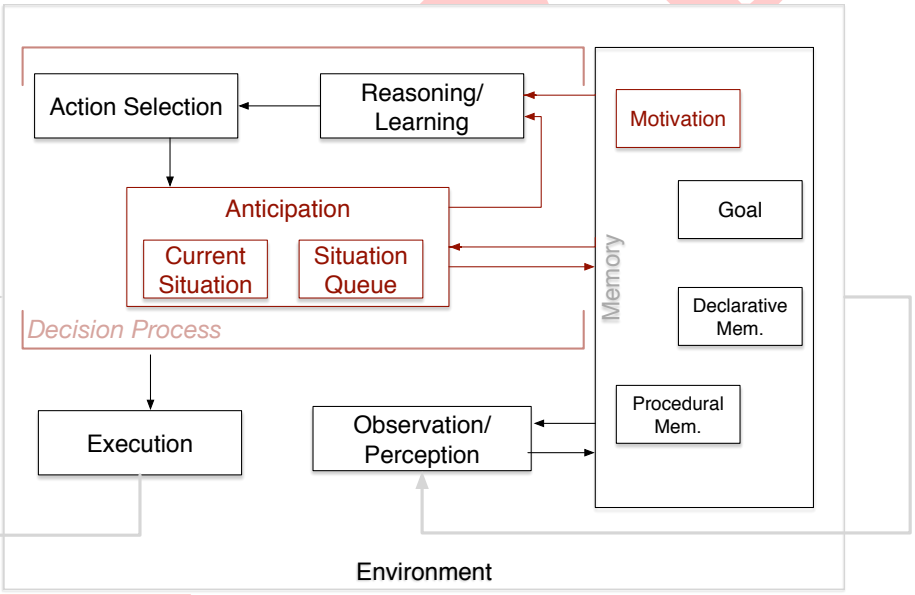

Figure 2: The Cognitive Architecture model that includes self-modeling ability for the decision-process.

R decomposes the cognition process in five specialized modules and shows how to integrate the modules in order to create a complete cognitive process. SOAR is based on a cyclic process that includes the production process and the decision one. One decision cycle follows each production cycle. Sigma blends elements both from ACT-R and SOAR and provides just a single long-term memory. We also studied other cognitive architectures [19][15][20][21]. The result was, starting from the standard model in Fig. 1 and the influence of all the other architectures, the architecture shown in Fig. 2. The figure recalls the deliberative behavior of an agent that perceives, reasons and acts. This architecture may be implemented, at the system level, by using Jason agents and their reasoning cycles[1].

Both this standard model and Jason, however, do not account the internal states, the fact that the environment may continually change in a not prescribed way and that the representation of the agents' environment also includes own inner world. To face this problem, we extended the higher level of the standard model, without modifying its core cycle.

Fig. 2 shows an extended version of the perception-action cycle in which we added modules for handling decision process and memory in the most useful way for our purposes. It can be seen that

[1] Details about Jason reasoning cycle are given in section 4.

a cognitive agent uses, for deciding which action to execute, inputs coming from the environment perception and from the memory. Agent decides actions to perform after a reasoning process and then it executes and continuously observes the results of its action on the environment.

This part is perfectly in line with existing cognitivist approaches. In order to integrate self-modeling and theory of mind abilities that, as already said, we guess to be fundamental for triggering the decision process in a way useful for human-agent teaming interactions, we needed to shift to a kind of emergent approach and we added some elements in the decision and memory modules.

We decided to represent knowledge not only including all the objects in the environment but also including goals to be pursued and motivations for executing a specific action. This allows us to consider the new perspective on the environment we introduced before: the environment is seen as something composed of objects, cognitive agents and all that is inside each agent. This latter elements, such as for instance the awareness to be able to do something, constitutes agent's self-model and then trigger the decision process along with the whole available knowledge on the "static" environment. Continuous observing and perceiving allow to constantly update and increase knowledge even during execution phase. In addition, in the ANTICIPATION module, we consider the possibility of creating an anticipation of an action result, some kind of post-condition on the state of affairs (the whole environment) at the end of each action. This part is fundamental for realizing human-like reasoning and decision process since it allows to anticipate also all other cognitive agents' behaviors and actions. In so doing, we may implement the elements of the theory of mind letting agents coordinating, cooperating and delegating actions at runtime and in a totally autonomous fashion. In the ANTICIPATION module we also included the elements for generating the CURRENT SITUATION and a QUEUE of possible situations, generated starting from the knowledge base and gained when the current situation is not applicable. The module, we simply named MOTIVATION, includes all the elements related to a human-like mental state; these elements have to be considered during the decision process. Examples of some of these elements are the emotional state, the level of trust in the other and in itself. Motivations, along with knowledge on goals, environment, including the self and own capabilities lead to take purposeful decisions.

In summary, in our architecture, the design process has the same input of the standard model but it also has an intermediate part. Executing an action has effects on the environment and also on the internal state. The perception refers to the external and internal world and also to the working memory, in our case goals and motivations.The working memory, so as the reasoning/learning process and action selection, is affected by the ability to generate anticipation.

In the following, we explain how we extended the Jason reasoning cycle to implement this architecture.

4 The BDI agent and its implementation by using Jason

Due to its features, the BDI agent-oriented paradigm well fits our need to model and implement the modules of the cognitive architecture shown in the previous section. Indeed, *Belief-Desire-Intention model* has been recognized as the most useful paradigm to implement human-like agents. We can talk about computer programs as cognitive agents owning a *mental state*. So, an agent is characterized by beliefs, desires and intentions:

- beliefs - are information that the agent has about the surrounding area or the world in general.
- desires - are all the possible *states of affairs* that an agent perform. The desire is not a *must to do action* but it could be seen as a condition that influences other actions.
- intentions - are the *states of affairs* that an agent decides really to perform. The intentions can be normally intended as a particular operation that other can be delegated to the agent or agent's consideration. This latter definition comes from practical reasoning systems inside agent.

Moreover, the BDI paradigm has its natural equivalent in the agent programming language: Jason [2]. Jason is an implementation of AgentSpeak language [22] that somehow allows overcoming the old denotation of software. The software is no longer something that provides a service by an exact coding, and that depends heavily on the intervention of the user. In Jason's logic, a computer program is something that has the know-how and chooses actions to pursue a goal on behalf of the human

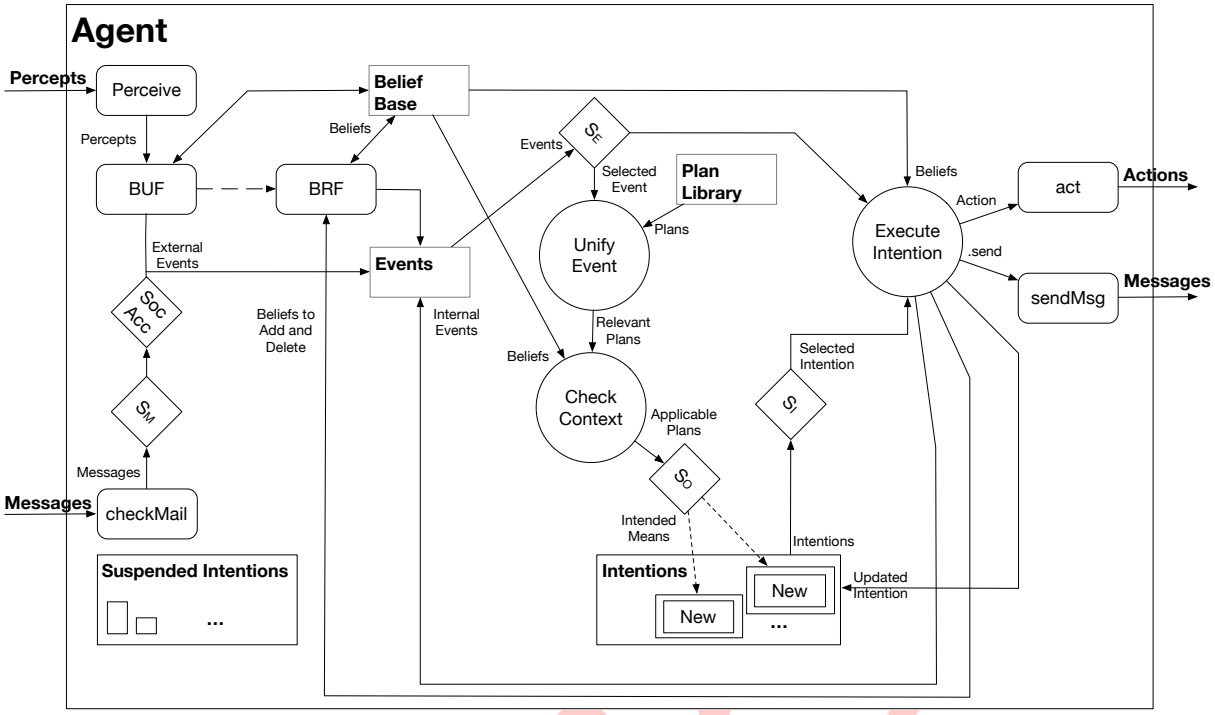

Figure 3: The original reasoning cycle of Jason Language - redrawn from [2].

and without his intervention. For this reason, a Jason program is called Agent. It does not yet have the characteristics to perfectly replicate how a human acts, but it can autonomously process the knowledge it possesses about how to do things. The basic idea behind Jason is to define what is called the program's know-how in the form of a set of plans. The Jason platform allows executing the deliberation process of a BDI agent that leads to choosing the intention to pursue within a set of possible states of affairs.

An agent can decide what to do, the set of actions in its repertoire to be undertaken starting from a set of data obtained through sensing and to modify the surrounding environment. In AgentSpeak, and therefore in Jason, deciding what to do means manipulating plans and the environment. Typically, a Jason agent has partial control over the environment in which it lives because it is also populated by other agents having control over their part of the environment. It can autonomously work because it is structurally defined to do this but cannot adapt itself in a dynamic environment; especially if the dynamicity of the environment derives from interactions with humans and other agents. The procedure for handling agent-agent interaction is standardized and mainly established at design time. Human-agent interactions have to be still explored, especially in the context of cooperation between humans and agents which presupposes delegation and/or selection of actions to be undertaken even by observing the human actions and skills. Jason is an optimum language to implement the standard model with the perception-action cycle. Fig. 3 represents the Jason agent's reasoning cycle and its modules realizing the beforesaid cycle; however it lacks means for implementing the architecture in Fig. 2. The agent reasoning cycle consists of three main processes, (i) a knowledge updater, (ii) an event handler and (iii) a module to act.

According to [2], *rectangles* represent the principal components that determine the agent's state, such as belief base component and all the components that handle the set of events, plan library and intentions. *Diamonds*, *circles* and *rounded boxes* are used for describing the functions used in the reasoning cycle. In particular, *circles* model the application processes and *diamonds* represent the selection functions. Jason allows to modify and customize the functions represented by *round boxes* and *diamonds*.

In the following section, we give a detailed description of the Jason reasoning cycle and then in section 5 we explain where and how we added new modules.

4.1 The Jason Reasoning Cycle

The main concepts in the Jason reasoning cycle are illustrated and described in the following table (Table 1). This subsection, Table 1 and Table 2 are loosely based on chapter 4 in the book by Bordini et al. [2].

Table 1: A complete summary of Jason's components and elements.

Agent	An agent is an entity with several capabilities. An agent is able to perceive and act in environment, communicate with others and reason about possible events. Agents have several skills and offer services.
Belief	Beliefs are information about the world.
Belief Base	A Belief Base is the structure where all beliefs are organized.
Plan	A Plan is composed by three parts: the triggering event, the context, and the body. The body contains other plans or actions.
Plan Library	A Plan Library is where all plans are stored and lets agent choosing which plan is more applicable or not to reach the goal.
Event	An event is a couple where the first component denotes the change that are taking place, and the second is the associated intention. It may be internal or external, the first is related to the goal the second is related to environment's changes.
Intention	Intentions are the states of affairs that the agent has decided to commit
Percepts	Percepts are referred to state of affairs in the surrounding world.
Context	As mentioned in [23] a context is the place where agents take into account others and/or where the others act to realize tasks.

In an AgentSpeak program, we define an agent type where we can set the initial state of beliefs, the events and the set of the plans that an agent could execute during its life-cycle.

The first thing that an agent does at the beginning of each reasoning cycle is perceiving the environment through its senses. This operation involves the belief base and the related function[2]. Beliefs are represented by using a symbolic form, the architecture has an internal *Literal* component that converts perceptions into a list of Literals; each of these is a single *percept* and this is a symbolic representation of a specific property perceived from the environment. This *percept* is detected by *perceive method* that implements the process. Once the agent has perceived the environment, it *updates* its internal belief base to reflect changes in the environment. The method able to do this is the *belief update function*, also known as *buf*. The Jason's *buf* presumes that every perceptible thing could be included into the list of percepts generated by the perceive module.

Generally, the *buf method* updates the belief base in a simple way; this method consists of two points: (i) each literal l in p not currently in b is added to b; (ii) each literal l in b no longer in p is deleted from b; where the symbol p is the set of current percepts and b is the set of literals in the belief base that the agent obtained from the last sensing process.

An *event* in Jason is considered as a couple, where the first component is the change occurred and the second is the associated intention. Furthermore, the event could be divided mainly into two categories, *internal* and *external events*. Each change produced by *buf method* invokes an *external*

[2]During the description of the reasoning cycle, we refer to Fig. 4 for the sequence of agent's activities and Fig. 5 for the related implemented classes. In Fig. 5 both classes of the original cycle and those of the extended one (in green) are represented.

Table 2: Description of functions for the Jason reasoning cycle.

Function	Description
perceive	The perceive method lets agents sense the environment and retrieve from it information about things. It implements the perceive function in Fig. 3.
socAcc	Social Acceptance Function establishes the reliability of other agents, it implements the *socAcc* function in Fig. 3.
selectEvent	Selects the events that will be handled in the reasoning cycle; it implements the $\mathscr{S}_{\mathscr{E}}$ function in Fig. 3.
selectIntention	Selects an Intention to be further executed in the reasoning cycle; it implements the $\mathscr{S}_{\mathscr{I}}$ function in Fig. 3. The default implementation executes intentions with a *round robin* scheduling process.
selectOption	This method is used to select one among several options (an applicable plan and an unification) to handle an event. It implements the $\mathscr{S}_{\mathscr{O}}$ function in Fig. 3.
selectMessage	Selects the message that will be handled in the current reasoning cycle; it implements the $\mathscr{S}_{\mathscr{M}}$ function in Fig. 3.
buf	Updates the belief base with the given percepts and adds all changes that were actually carried out as new events in the set of events.
brf	Revises the belief base with a literal to be added (if any), a literal to be deleted (if any), and the Intention structure that required the belief change.
act	Act function lets agents execute action in the environment.

event. This kind of event is created when the source of belief is related to the percept. In this case, the associated couple contains as the first member the change produced and as the second member an *empty intention* denoted by a $\top$.

Another important module is the communication system. Inside the cycle, an agent could retrieve information about the environment or other agents through others in the same system. At this point, the cycle checks for new messages that might have been sent from others. This distribution system is just integrated into Jason and it works such a mailbox for each agent.

The method within the reasoning cycle is called *checkMail* that simply checks for received messages and makes them available at the level of AgentSpeak interpreter to be handled by agents.

The *message selection function*, denoted by $\mathscr{S}_{\mathscr{M}}$ in Fig. 3, selects the first message in the list in the default implementation. By security reason a *social acceptance function* is defined into the system, the aim is ignoring potential malicious attack that others could do or bypass wrong messages that could spoil the reasoning loop, the default implementation accepts all messages from all agents.

A relevant role is taken by events since they represent the effective changes in the environment or agent's own goal. The principal cognitive skill of the BDI Jason Agent is handling events, this happens only managing them one at time; for instance in a hypothetic cooperative scenario, the agent could have the necessity to evaluate other events before the first in the set of events; to solve this problem Jason lets the user customize a specific function called *event selection function* denoted with $\mathscr{S}_{\varepsilon}$ in Fig. 3. The default implementation selects the first event in the list of events; it works like a FIFO structure if not customized.

Once relevant event has been selected, the agent needs to select a set of reliable plans from a collection called *Plan Library* that will permit to react to a specific event with a designed action. The agent searches into the *library* all necessary plans for each event. After selecting the relevant plans, a unification process helps the reasoning cycle to catch a set of relevant plans on which a check context process will be applied. The output of the latter process lets deleting all plans that do not match with the current context and only a subset of all relevant plans will be marked as *applicable plans*.

Given the set of applicable plans and all the knowledge acquired by agent through perception and

communication skills, converted into a set of literals represented as beliefs, the agent is able to choose one of the selected applicable plans through an internal method: *Option Selection Function* denoted with the symbol $\mathscr{S}_{\mathscr{O}}$ in the Fig. 3. The output of $\mathscr{S}_{\mathscr{O}}$ produces what is called *intended means* and this name is associated to the mean that the agent intends to execute to handle the related event. It is worth to remind that each plan in the set of applicable plans represents an alternative to reach the goal. The default implementation of this function considers as applicable plan the first in order of appearance in the plan library; the position of each plan is defined in the agent definition file written in AgentSpeak. Thus in the default implementation, the agent attempts to choose the applicable plan according to the agent's developer.

To select which intention to be computed during the cycle, a *intention selection function* denoted with the symbol $\mathscr{S}_{\mathscr{I}}$ is used. $\mathscr{S}_{\mathscr{I}}$ has a default implementation like a *round-robin scheduler*. Moreover every times an intention is extracted, it is removed from the list of intention after the execution and when this will be inserted back into the list, this will be added at the end of the list. In this way every intention have the same portion of agent's attention.

The last stage of the reasoning cycle performs an action in the agent environment or produces a message to communicate with other agents. Generally, the action that an agent performs are (i) *environment action*, (ii)*achievement action*, (iii) *test goal*, (iv) *mental notes*, (v) *internal actions* or (vi) *expressions*.

5 The Jason Reasoning Cycle Extension

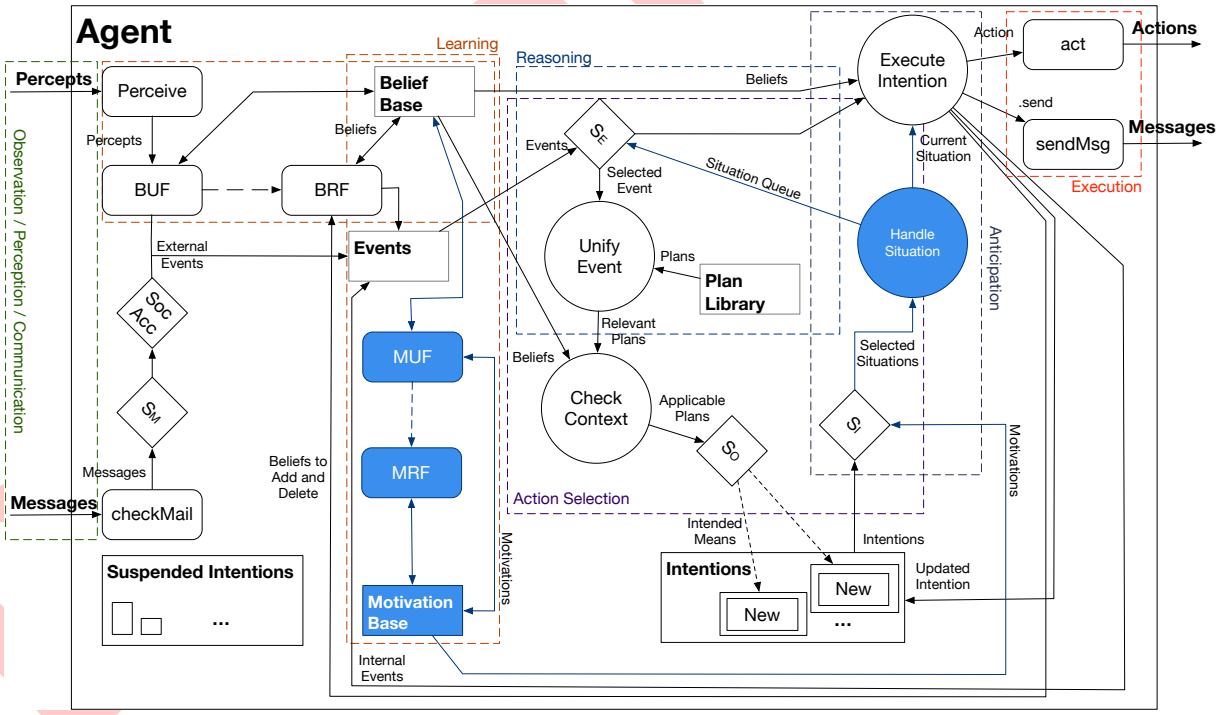

Figure 4: The extended reasoning cycle for Jason's agents

The reasoning cycle proposed in Fig. 3 as said before develops three general processes: the knowledge update process, the handle event and the acting module.

The elements: motivation, goal, anticipation (current situation and queue) are not handled by the traditional Jason reasoning cycle. In some way, goals are implicitly treated in the plan and the plan library plus the context and the intention. However, it is not enough for our objectives. We need a structure to match goals with the anticipated results of single actions. This provides a cornerstone for realizing self-modeling.

Motivation serves for modeling all that is related to the internal state, the inner world. It is a kind of belief (the state of the environment) but they differ from a conceptual point of view. In fact, beliefs are data about the environment, are values providing pre-conditions for a plan to be activated or selected.

Table 3: Definitions new elements integrated in Jason.

Component	Description
Motivation	A Motivation is a form of belief for self-awareness modeling. Motivations are information that the agent has about itself. Motivation contains knowledge about the consciousness of the agent.
Motivation Base	A Motivation Base is where all motivations are organized.
Situation	A Situation is an extension of *Intention*. It represents the future state of affairs after the execution of agent's action.

Motivation is, in some senses, a superstructure of beliefs including the agent's mental state.

Anticipation, in the two forms, current situation and situation queue, is used for generating a sort of simulation of the scene, the state of the world, if everything would go well. During the reasoning cycle, the agent decides which action to commit also after having evaluated the anticipation against the first component of the event, thus reinforcing the decision process.

In Table 4 and 3 a summary of definition of concepts used in the extended cycle (Fig. 4) is reported.

Jason reasoning cycle does not allow to implement these elements, above all the anticipation so, in order to add them, we created two new functions MUF and MRF with their related methods muf and mrf in the agent class (Fig. 5), one other principal component, the "Motivation Base" and one new application process, the "Handle Situation". We also modified some selection functions as will be illustrated later in this section. Through motivations, it is possible to solve other complex plans and to force to select an intention to accomplish agent's desires. The deliberative process of actions is not limited to simple execution of a plan assigned to agents at design time and stored into the Plan Library, but the extension provides a valid alternative built on the motivation base. With this knowledge, the agent tries to define new alternative *applicable plans* that should be successful plans cause these are generated by checking the status of the agent internal state and the context.

Anticipation, as said before, has the aim to produce a *queue of situations* where the head of the queue is the situation (alias intention) that the cycle is scheduling to handle for acting. The tail of the queue contains all possible future situations in according to the inner state of the agent including its motivation. The mechanism of the anticipation module recalls the perception loop described in [24]. Another important module that we add to the reasoning cycle is motivation.

The reasoning process starts with a *perceive method*. In Fig. 4 percept is handled by a *perceive function*, this latter handles two kinds of percept, the first is external perception or better, percepts that come from the environment and the second is the internal perception. In this last, the agent sense itself looking for features such as for instance execution time, stress-level, emotive state and other motivational features. This is the first significant difference with the original cycle. Inner perceptions are not defined as normal beliefs (*Literal*) but once agent perceives the feature this is converted into motivation. Each motivation is organized in a MotivationBase that is an extension of the BeliefBase (see in Fig. 5). In the same UML diagram, it is possible to find some changes that we did to implement the cycle. As described we extended the *AgArch class* by creating another derived class called *AgArchMotivated* that implements the handling of internal perceptions. Knowledge is updated by means of *belief update function* and *motivation update function*. As said for the *buf method* in the original Jason reasoning cycle, the operating mechanism is not modified but we added a new branch to handle the inner state.

Initially every perception acquired is a belief with some operations the belief update function and the belief revision function convert internal perceptions in motivations and using the *motivation update function*, the system saves internal perceptions (aka motivations) in the *MotivationBase*. Processing motivations involves also motivation revision function to revise the MotivationBase. These functions are implemented in *AgentMotivated* class, this class extend the Jason *Agent* class. Information about the state of affairs of the environment is also perceived by the communication module. The reasoning contains methods to communicate with other agents.

For this purpose, Jason has two important functions to handle the communication between agents. The *checkMail method* is the first function that the agent does as shown in Fig. 4 and it gets external messages sent by agents and organizes them into a mailbox assigned to the agent. Here, the function

Table 4: Description of functions for the extended Jason reasoning cycle.

Function	Description
perceive	The perceive method lets agents sense the environment and retrieve from it information; in our approach, the function gets information about itself through sensing the status of internal parts and the status of the mind. It implements the perceive function in Fig. 4.
selectEvent	Our implementation takes in input a queue of events and motivations to select the plan that accomplish agent's desires (see Alg. 2).
selectIntention	Selects an Intention to be further executed in the current reasoning cycle; it implements the $\mathscr{S}_{\mathscr{I}}$ function in Fig. 4. Our implementation uses an intelligent scheduling that selects the right intention considering not only the queue of intentions but also a queue of situations that helps the algorithm to choice the better situation to compute in the handle situation. This function looks in future scenario to accomplish the desires and respects the agent's motivation (see Alg. 4).
muf	*Motivation update function* updates the motivation base with given perception.
mrf	*Motivation revision function* revises the motivation base with a literal to be added (if any), a literal to be deleted (if any).
buf, brf, $\mathscr{S}_{\mathscr{M}}$**, socAcc,** $\mathscr{S}_{\mathscr{O}}$	These functions are not modified. For a description read the Table 2.

remains the same of the original cycle (*AgArch* class in Fig. 5). The *social acceptance function*(aka *socAcc*) and its associated method are modified for our purposes and their functionalities are described in Table 2. After beliefs and motivations are updated, new events are generated. At this point, in the original cycle the *selectEvent* function, given as parameter a queue of events, returns the related *poll function*. Algorithm 1 shows as the original *selectEvent* function works. Algorithm 2 shows how we modified the *selectEvent* function in order to consider the situation.

Algorithm 1 The *selectEvent* algorithmn implemented in Jason.

```
1: procedure S_E(Queue<Event> events)
2:     queue← events
3:     return queue.poll()
4: end procedure
```

So $\mathscr{S}_{\mathscr{E}}$ function implements an algorithm that, given as parameters the event queue and the situation queue generated at the previous cycle, executes a *generateEventsQueue function* to obtain a queue of events that are reformulated considering the queue of possible future scenarios carried by the queue of situations (if any). Once the algorithm has generated the *Selected Event* as shown in Fig. 4, such a result of the $diamond\ \mathscr{S}_{\mathscr{E}}$ function, a *unification process* is fired. It unifies events evaluating the previous results and a list of plans generated by a *plan library* module. This process is exactly the one described in the previous section 4. The result of this process is a list of relevant plans. Once plans are selected another process is started.

The following process *check context* (Fig. 4) is the same of the original cycle. This process is not influenced by the *motivation module* or *anticipation module* because the relative check context for our scope is included in the *select event function* with the *generateEventQueue* function as you can see in algorithm 2. After the context is verified, the process generates *applicable plans* that are given as input to the function *applicable plan selection function* denoted as $\mathscr{S}_{\mathscr{O}}$ in Fig. 4. The output of this

Algorithm 2 The *selectEvent* algorithmn implemented in the extended Jason reasoning cycle.

```
1: procedure S_E(Queue<Event> events, Queue<Situation> situations)
2:     Queue<Event> eventsQueue ← events
3:     Queue<Situation> situationsQueue ← situations
4:     Queue<Event> queue := generateEventsQueue(eventsQueue,eventsSituations)
5:     Event selectedEvent := queue.poll()
6:     return selectedEvent
7: end procedure
```

function is called *intended means* that is the chosen applicable plan because the actions executed by that plan is the means that the agent intends to execute to handle the chosen event. The intended means is converted into intentions and will be handled by the *intention select function*. The list of intentions, this is given as argument for the *intention select function* and through a handle situation process our reasoning cycle extracts from this latter process the current situation, that is shown in the UML diagram a derived class of Intention. The original *intention selection function* $\mathscr{S}_{\mathscr{I}}$ implements a poll function as argument as described in algorithm 3. Our implementation, instead, involves two

Algorithm 3 The *selectIntention* algorithm implemented in Jason.

```
1: procedure S_I(Queue<Intention> intentions)
2:     queue← intentions
3:     return queue.poll()
4: end procedure
```

functions (Algorithm 4). The first creates a simulation process that implements a structure similar to the one described in [24] and the second verify the queue of situations generated at the step before. In algorithm 4, it is proposed our pseudo-code for the reasoning cycle.

Algorithm 4 The *selectIntention* algorithm implemented in the extended Jason reasoning cycle.

```
1: procedure S_I(Queue<Intention> intentions, Queue<Motivation> motivations)
2:     intentionsQueue ← intentions
3:     motivationsQueue ← motivations
4:     Queue<Situation> situations := simulate(intentionsQueue,motivationsQueue)
5:     Queue<Situation> selectedSituations := verify(situations)
6:     return selectedSituations
7: end procedure
```

Regarding code, the situation queue is internal to the agent so the code structure is not changed because a poll function is always called on the result of the previous function. So the execute intention process has as input a situation that is an intention as evidenced in the UML diagram in Fig. 5. This process tries to perform all actions included in the current situation and the agent is able to produce the *act* function, after which the reasoning cycle could be restart.

6 Discussion and Conclusions

The work presented in this chapter deals with systems (such as human-agent interaction systems) working in not completely known operational conditions. This research area lacks techniques and tools for modeling and implementing all that regards the decision-process activities of systems intrinsically self-adaptive and autonomous.

We present the implementation of agent's decision making process in a dynamic context. Our proposal is based on the fact that agent's decision-making-process is determined by processing data coming from observation of the external environment and also by the knowledge the agent has about itself and other agents acting around. The implementation of such a system is a hard task because its features can be seen only at runtime, during the interaction with the whole environment. Therefore, the system must be treated and implemented with self-adaptive characteristics.

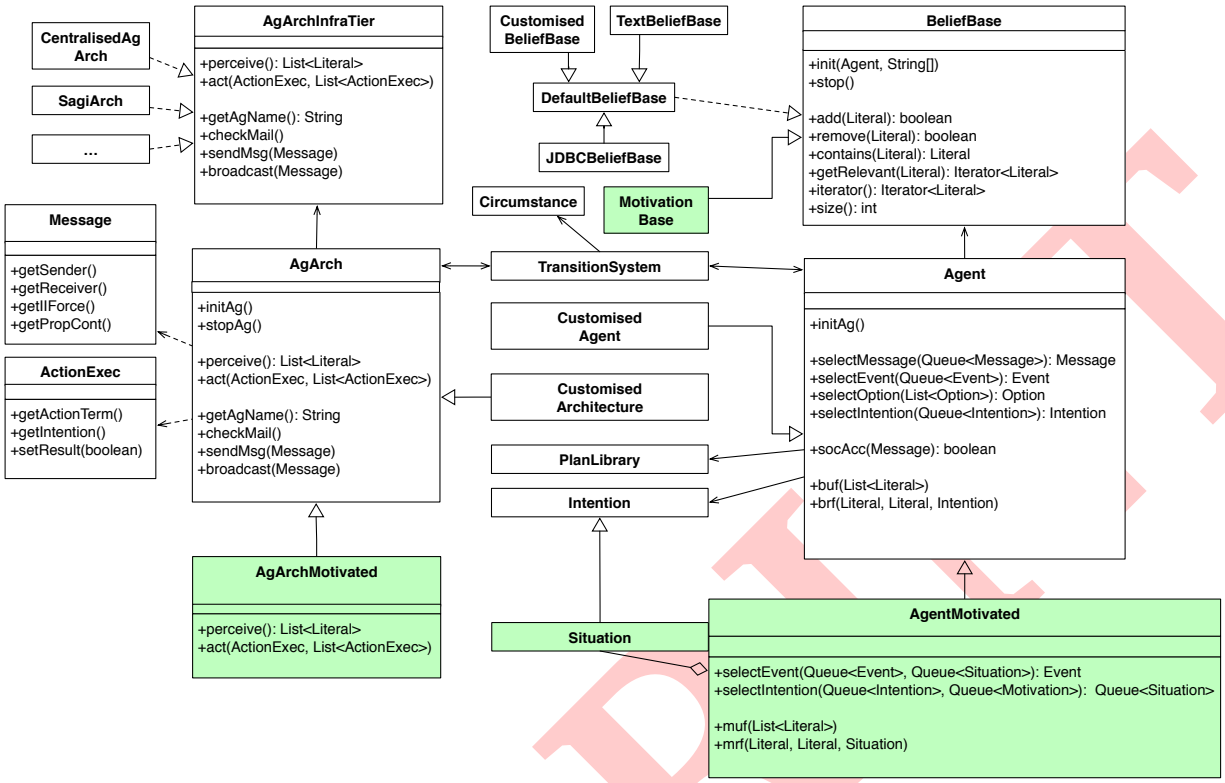

Figure 5: Agent and Agent Architecture Class Diagram and the related extensions for implementing the new reasoning cycle.

We have exploited the power of BDI agents and the Jason language, which natively allow creating agents that perform a deliberation and means-ends reasoning process. We modified the Jason reasoning cycle to include modules to manage events, plans, and intentions selection in order to take into account what we call motivations in addition to traditional beliefs. To complete the infrastructure, even at the agent coding level, we modified classes of the Jason component called *user-defined*. In particular, we added the classes needed to implement the part of the new reasoning cycle by adding the methods necessary for the agent to be able to choose the plan to pursue; in this way agents use a cognitive process based on what we called motivations that embody the mental states of the agent.

In order to model and implement motivations, we could have extended the belief concept instead of creating a new element. We decided to separate the two for two reasons: first, we want to resemble the human mind where these two aspects are considered separately, indeed the observation of the state of the environment brings to computation on data whereas elaboration on mental state gives an agent the attitude to make propositions such as believing, hating, loving and so on. The second reason, the most important for us, is that the continuation of this work will regard the modification of the Jason interpreter. We think that it will be more efficient t work in a modular fashion and adding elements to the interpreter instead of extending the existing one so to maintain a high degree of modularity and a low coupling.

It is worth to note that the proposed cycle extension does not alter the original Jason agent reasoning at a high level, but extends its capabilities, allowing the development of agents able to manage at the same time the sense of self and the theory of the mind together with the usual decision-making process. This work is the initial part of a larger project for the implementation of complex adaptive systems including knowledge update, selection and creation of new plans at runtime. The approach we are suggesting has given some good results in the validation phase during a series of experiments conducted in the robotics laboratory of the University of Palermo [25][26][27]. In the future, we think to definitively formalize the approach with the addition of all the design aspects to experiment with a complex case study.

Acknowledgment. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0232.

References

[1] M. Georgeff and A. Rao, "Rational software agents: from theory to practice," in *Agent technology*. Springer, 1998, pp. 139–160.

[2] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007, vol. 8.

[3] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with jacamo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747–761, 2013.

[4] J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel, "Software engineering processes for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 51–75.

[5] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The knowledge engineering review*, vol. 10, no. 2, pp. 115–152, 1995.

[6] A. S. Rao, M. P. Georgeff *et al.*, "BDI agents: from theory to practice." in *ICMAS*, vol. 95, 1995, pp. 312–319.

[7] B. H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*. Springer, 2009, pp. 1–26.

[8] N. A. Stillings, C. H. Chase, and M. H. Feinstein, *Cognitive science: An introduction*. MIT press, 1995.

[9] A. Clark, *Mindware: An introduction to the philosophy of cognitive science*. Oxford University Press, 2000.

[10] J. S. Kelso, *Dynamic patterns: The self-organization of brain and behavior*. MIT press, 1997.

[11] L. Baresi and C. Ghezzi, "The disappearing boundary between development-time and run-time," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 17–22.

[12] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, Oct 2009.

[13] R. Sun, "The importance of cognitive architectures: An analysis based on clarion," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 19, no. 2, pp. 159–193, 2007.

[14] J. E. Laird, A. Newell, and P. S. Rosenbloom, "Soar: An architecture for general intelligence," *Artificial intelligence*, vol. 33, no. 1, pp. 1–64, 1987.

[15] S. Franklin, T. Madl, S. D'Mello, and J. Snaider, "Lida: A systems-level architecture for cognition, emotion, and learning," *IEEE Transactions on Autonomous Mental Development*, vol. 6, no. 1, pp. 19–41, 2014.

[16] J. E. Laird, C. Lebiere, and P. S. Rosenbloom, "A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics." *AI Magazine*, vol. 38, no. 4, 2017.

[17] J. R. Anderson, M. Matessa, and C. Lebiere, "Act-r: A theory of higher level cognition and its relation to visual attention," *Human-Computer Interaction*, vol. 12, no. 4, pp. 439–462, 1997.

[18] D. Koller, N. Friedman, and F. Bach, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[19] D. E. Kieras and D. E. Meyer, "An overview of the epic architecture for cognition and performance with application to human-computer interaction," *Human-computer interaction*, vol. 12, no. 4, pp. 391–438, 1997.

[20] W. D. Christensen, C. A. Hooker *et al.*, "Representation and the meaning of life," *Representation in mind: New approaches to mental representation*, pp. 41–69, 2004.

[21] M. Shanahan and B. Baars, "Applying global workspace theory to the frame problem," *Cognition*, vol. 98, no. 2, pp. 157–176, 2005.

[22] A. S. Rao, "Agentspeak (l): BDI agents speak out in a logical computable language," in *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer, 1996, pp. 42–55.

[23] C. Castelfranchi and R. Falcone, *Trust theory: A socio-cognitive and computational model*. John Wiley & Sons, 2010, vol. 18.

[24] V. Seidita and M. Cossentino, "From modeling to implementing the perception loop in self-conscious systems," *International Journal of Machine Consciousness*, vol. 2, no. 02, pp. 289–306, 2010.

[25] A. Chella, F. Lanza, and V. Seidita, "Representing and developing knowledge using Jason, CArtAgO and OWL," in *Proceedings of the 19th Workshop "From Objects to Agents", Palermo, Italy, June 28-29, 2018.*, 2018, pp. 147–152.

[26] A. Chella, F. Lanza, A. Pipitone, and V. Seidita, "Knowledge acquisition through introspection in human-robot cooperation," *Biologically Inspired Cognitive Architectures*, vol. 25, pp. 1–7, 2018.

[27] A. Chella, F. Lanza, and V. Seidita, "A cognitive architecture for human-robot teaming interaction," in *Proceedings of the 6th International Workshop on Artificial Intelligence and Cognition*, Palermo, July 2-4 2018.